# Architecture systolique pour la correction automatique de libelle d'adresse

Dominique Lavenier, Jean-Luc Scharbarg, Patrice Frison

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 995

*Programme 2*

# ARCHITECTURE SYSTOLIQUE POUR LA CORRECTION AUTOMATIQUE DE LIBELLE D'ADRESSE

**Dominique LAVENIER**
**Jean-Luc SCHARBARG**
**Patrice FRISON**

**Mars 1989**

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex: UNIRISA 950 473 F
Télécopie: 99 38 38 32

# ARCHITECTURE SYSTOLIQUE POUR LA CORRECTION AUTOMATIQUE DE LIBELLE D'ADRESSE

# SYSTOLIC ARCHITECTURE FOR AUTOMATIC ADDRESS CORRECTION

Dominique Lavenier, Jean-Luc Scharbarg, Patrice Frison

24 janvier 1989

**Publication Interne n° 455 - Février 1989 - 22 Pages**

**Résumé**

Le tri automatique du courrier est divisé en plusieurs étapes. Une des premières opérations passe par la lecture optique de l'adresse. Les mots reconnus comportent des erreurs dûes à la fois aux limites du lecteur optique et aux adresses mal orthographiées. Il est donc nécessaire, lorsqu'un mot à été mal reconnu, de rechercher parmi un vocabulaire donné les mots qui lui "ressemblent" le plus. Les techniques de programmation dynamique sont particulièrement efficaces pour traiter ce type de problème. Cependant, lorsque de gros dictionnaires ou d'importantes bases de données sont mis en jeu, l'implémentation sur des processeurs traditionnels devient inadaptée. En effet, les très bons résultats obtenus avec ces méthodes se paient par un temps de calcul prohibitif incompatible avec les contraintes de temps imposées sur le traitement global d'une adresse. Par contre, leur parallélisation sur des architectures de type systolique semble être une voie particulièrement prometteuse. Cet article présente, d'une part, un algorithme issu du concept de programmation dynamique et optimisé à partir de données réelles et, d'autre part, les mises en oeuvre possibles sur des architectures systoliques différentes.

**Abstract**

Automatic mail sorting consists of several stages. The first one is the address optical reading. Some of the recognized words contain optical or spelling errors. These words need to be searched in a vocabulary in order to find the most probable correct words. Dynamic programming techniques using Levenshtein metric concept are particularly efficient for string correction. However, the implementation of the method on conventional computer is inadequate when large vocabularies or data base environment are required. Indeed, the method is very time consuming and cannot cope with address recognition timing constraints. Fortunately, dynamic programming methods are good candidates for parallel implementation and systolic architectures seem to be an interesting structure. This paper presents first an algorithm based on dynamic programming concept which has been optimised on real data. It then describes implementations on several systolic architectures.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

# 1 Introduction

L'utilisation croissante de données alphanumériques dans les systèmes informatiques a rendu très importante la détection et la correction de fautes d'orthographe [Hall80]. De nombreux algorithmes ont été proposés et utilisent différentes techniques pour corriger ou retrouver des informations erronées.

La méthode des abréviations [Blai60], [Davi62], [Dame64], [Durh83], [Bick87], consiste à trouver une séquence de caractères courte et significative pour coder chaque mot. Les abréviations sont construites à partir de différents critères dépendant de l'application envisagée. Cette méthode est en général inutilisable pour les gros vocabulaires.

Une méthode basée sur l'analyse des fréquences des digrammes et des trigrammes (groupes de deux ou trois lettres) [Morr75], [Ange83], donne de bons résultats sur de gros vocabulaires, mais ne détecte pas toutes les erreurs. Des méthodes probabilistes ont également été explorées par [Kash84].

Les approches les plus prometteuses utilisent le concept de métrique de Levenshtein [Hall80]. La distance de Levenshtein entre deux chaînes de caractères est définie comme le nombre minimum d'opérations d'édition (insertions, omissions, substitutions) nécessaires pour passer de la première chaîne à la deuxième. Elle a été présentée par Wagner et Fisher dès 1974 [Wagn74], [Lowr75] et a influencé de nombreux autres chercheurs [Okud76], [Vero87]. Les principaux avantages de cette méthode sont sa généralité, sa flexibilité et les bons résultats qu'elle procure. De plus, la distance de Levenshtein peut être calculée efficacement en utilisant les méthodes de programmation dynamique.

La correction d'un libellé d'enveloppe dactylographié doit tenir compte de deux types d'erreurs, les erreurs commises pendant la lecture optique et celles issues d'adresses mal orthographiées.

Comme ces erreurs ne peuvent pas a priori être différenciées, une même méthode de correction doit être appliquée. Une correction basée sur le concept de métrique de Levenshtein se révèle être une technique appropriée à ces deux types d'erreurs.

Ce processus de correction s'intègre dans une unité de reconnaissance d'adresses actuellement à l'étude. Le tri du courrier constitue un problème complexe, tant du point de vue qualité que du point de vue rapidité. Cette opération est réalisée par une unité de reconnaissance d'adresses qui, pour chaque enveloppe, doit, soit l'indexer (la coder en acheminement et en distribution), soit la rejeter. Le but est d'indexer correctement le plus de lettres possible. Des machines de tri automatique du courrier dactylographié existent déjà. Elles sont toutefois assez rudimentaires (pour des problèmes de rapidité). Elles ne reconnaissent que les adresses ayant une syntaxe standard. Les correcteurs de chaînes de caractères utilisés autorisent au plus une insertion ou une omission et les coûts des opérations d'édition sont entièrement basés sur des critères optiques. Les fautes d'orthographe, même usuelles, sont donc très pénalisantes. La machine actuellement à l'étude permet de traiter des syntaxes d'adresses variées. Le processus de correction de chaînes de caractères qui s'y intègre utilise la distance de Levenshtein. Il autorise donc un nombre a

priori illimité d'insertions et d'omissions, et prend en compte les fréquences d'apparition des différentes erreurs dans la réalité.

Nous présentons d'abord brièvement le fonctionnement interne de la machine visée, et la place occupée par le correcteur de chaînes. Nous détaillons ensuite l'algorithme de correction, en précisant une mise en oeuvre séquentielle optimisée et les taux de reconnaissance obtenus. Puis nous exposons une mise en oeuvre parallèle possible, en l'occurence sur des architectures systoliques, et proposons diverses solutions (architectures et algorithmes) ainsi que les performances temporelles attendues.

# 2 Le Tri Automatique du Courrier

Le tri automatique du courrier est réalisé par une unité de reconnaissance d'adresses. Le traitement comporte plusieurs phases :

- une reconnaissance optique génère une image numérisée de l'enveloppe,
- une reconnaissance caractères recherche les lignes dans l'image, découpe les lignes en mots, puis les mots en caractères,
- une analyse contextuelle réalise l'analyse syntaxique et sémantique de l'adresse.

L'analyse contextuelle détermine la destination de l'enveloppe. Pour cela, il faut corriger les mots du libellé pour pouvoir traiter correctement l'enveloppe. Chaque mot est donc envoyé à un serveur de dictionnaire qui détermine les mots les plus proches. Les différents dictionnaires utilisés sont :

- le dictionnaire des communes, comprenant environ 30.000 mots,
- les dictionnaires des noms de voies (un pour chaque grande ville). Le plus grand, correspondant à Paris, contient environ 6.000 mots,
- des dictionnaires spéciaux, contenant des listes de mots comme CEDEX ou BRUNE. Ces dictionnaires sont très petits. Ils n'excèdent jamais 50 mots.

L'objectif fixé pour le serveur de dictionnaire est de corriger au moins 90 % des mots. Il faut donc être capable de détecter tous les types de fautes (insertions, omissions, substitutions), n'importe où dans le mot. Il faut également pondérer correctement chaque erreur. C'est pourquoi la méthode basée sur la distance de Levenshtein est adaptée. Elle autorise en effet des erreurs n'importe où dans le mot, et les coûts des différentes opérations d'édition permettent de prendre en compte la fréquence d'apparition de chaque erreur.

L'objectif temporel est de traiter 15 enveloppes par seconde (déterminer son code d'acheminement et de distribution). La machine étant pipelinée, il faut indexer une enveloppe en 210 ms. Des évaluations ont montré que, sur ces 210 ms, jusqu'à 100 ms peuvent

être utilisées pour corriger les mots. Pour une enveloppe donnée, le serveur doit traiter au maximum 10 mots. Il faut donc corriger un mot en 10 ms.

Nous détaillons maintenant l'algorithme de correction de chaînes proposé pour le serveur.

# 3 L'algorithme de Correction de Chaînes

## 3.1 L'idée de base

L'algorithme doit comparer deux chaînes de caractères en calculant une distance d'édition, qui correspond au coup minimal pour transformer la première chaîne en la deuxième. La transformation se fait à l'aide d'opérations d'édition élémentaires, qui peuvent être :

- la substitution d'un caractère par un autre caractère,
- l'insertion d'un caractère,
- l'omission d'un caractère.

N'importe quelle chaîne de caractères peut être ainsi transformée en n'importe quelle autre chaîne. Le calcul s'effectue de la manière suivante : Soit la chaîne d'entrée issue de la reconnaissance caractères

$$\begin{aligned} X = \quad & ((x_{1,1}, s_{1,1}) \cdots (x_{1,p_1}, s_{1,p_1}), \\ & \cdots, \\ & (x_{i,1}, s_{i,1}) \cdots (x_{i,p_i}, s_{i,p_i}), \\ & \cdots, \\ & (x_{n,1}, s_{n,1}) \cdots (x_{n,p_n}, s_{n,p_n})) \end{aligned}$$

Pour le $k^{ième}$ caractère, $p_i$ choix $(x_{i,1} \cdots x_{i,p_i})$ sont proposés. A chaque choix $x_{i,k}$ est associé un indice de confiance $s_{i,k}$. Cet indice représente le degré de ressemblance entre le choix et la forme présente dans l'image numérisée de l'enveloppe.

Soit un mot de dictionnaire

$$Y = \quad (y_1, \cdots, y_j, \cdots, y_m)$$

Soit $d(y, x)$ , le coût de l'opération d'édition transformant $y$ en $x$, et $\delta$ , le caractère nul.

La distance $D(m, n)$ entre $Y$ et $X$ est donnée par la relation de récurrence suivante :

$$D(i,j) = \min \begin{cases} D(i-1, j-1) & + & \sum_{k=1}^{p_j} s_{jk} * d(y_i, x_{jk}) \\ D(i-1, j) & + & \sum_{k=1}^{p_j} s_{jk} * d(\delta, x_{jk}) \\ D(i, j-1) & + & d(y_i, \delta) \end{cases} \quad (1)$$

$d(y_i, x_{jk})$ est le coût de la substitution de $y_i$ par $x_{jk}$. $d(\delta, x_{jk})$ représente l'insertion de $x_{jk}$, tandis que $d(y_i, \delta)$ représente l'omission de $y_i$.

Le nombre de choix $(p_j)$ pour chaque caractère de la chaîne d'entrée peut être arbitrairement limité. Si cette limite est fixée à un, l'indice de confiance n'est pas pris en compte.

Les coûts des différentes opérations d'édition sont des entiers compris entre 0 et 100. Ils peuvent être, soit fixés arbitrairement, soit déterminés par un apprentissage sur des données réelles. La première solution ne nécessite aucun calcul préalable, mais les valeurs retenues ne sont pas nécessairement réalistes. La deuxième élimine cet inconvénient, à condition que l'échantillon d'apprentissage soit représentatif.

## 3.2 L'application de l'algorithme au problème des adresses

La particularité du problème permet de limiter la complexité de l'algorithme, essentiellement du point de vue du temps de calcul, mais aussi parfois du point de vue de l'espace mémoire utilisé. Ces diverses optimisations vont maintenant être exposées.

### 3.2.1 La réduction des tables de coûts

La reconnaissance caractères réalise une bonne segmentation des mots et des caractères. Les insertions et les omissions sont donc relativement rares. Un comptage sur de données réelles a montré qu'elles représentent à peine 2 % de l'ensemble des opérations d'édition. La faiblesse de ce chiffre permet de ramener les coûts des insertions et des omissions à deux coefficients. Le coefficient d'insertion est égal à la moyenne de tous les coûts d'insertion, et le coefficient d'omission est égal à la moyenne de tous les coûts d'omission. Cela permet de limiter la taille des tables de coûts, et donc de réduire l'espace mémoire utilisé.

### 3.2.2 La longueur des mots

Les insertions et les omissions étant relativement rares, il est judicieux de ne comparer une chaîne de longueur $n$ qu'avec les mots du dictionnaire dont la longueur est proche de $n$. Plus formellement, $r$ est définie comme étant la variation maximale de longueur autorisée. Une chaîne de longueur n est comparée avec les mots de longueur $l$, telle que $n - r \leq l \leq n + r$.

Sur le dictionnaire des communes, la réduction du nombre de mots à comparer est particulièrement importante pour les mots courts et les mots longs, assez peu nombreux. Elle l'est beaucoup moins sur les mots de longueur moyenne (7 à 9 caractères), qui constituent la majeure partie du dictionnaire.

Il est à noter que cette optimisation est plus efficacement mise en oeuvre si le dictionnaire est trié par taille de mot (Cela permet de localiser aisément la partie de dictionnaire à comparer avec la chaîne d'entrée).

### 3.2.3 Les diagonales

Cette optimisation s'inscrit dans le même ordre d'idée que la précédente. Il s'agit en effet de ne calculer que les *D(i,j)* tels que $i-r \leq j \leq i+r$. Cela signifie que le chemin optimum ne s'éloigne jamais plus de $r$ de la diagonale (d'où le nom donné à cette optimisation). Le nombre de *D(i,j)* à calculer est $n(2r+1) - r(r+1)$, où $n$ est la longueur de la chaîne d'entrée. Le gain le plus important est obtenu pour les mots les plus longs.

### 3.2.4 L'utilisation des préfixes

Dans un dictionnaire trié par longueur de mot et par ordre alphabétique, on constate que le calcul pour deux mots consécutifs commençant par les mêmes lettres (préfixe commun) est en partie identique. Ainsi, pour le dictionnaire des communes, le nombre de lettres de tous les préfixes communs représente 42 % du nombre de lettres total. Il faut éviter de refaire plusieurs fois les mêmes calculs. Soit $Y_1 = (y_{11}, \cdots, y_{1m})$ et $Y_2 = (y_{21}, \cdots, y_{2m})$, deux mots du dictionnaire consécutifs et de même longueur. Soit $C_{y_1,y_2}$, la longueur du préfixe commun à $Y_1$ et $Y_2$ : $0 \leq C_{y_1,y_2} \leq m-1$ telle que, pour $1 \leq k \leq C_{y_1,y_2}$, $y_{1k} = y_{2k}$.

Il est clair que, pour $0 \leq i \leq C_{y_1,y_2}$ et pour tout j, les *D(i,j)* sont identiques pour $Y_1$ et pour $Y_2$. Les calculs sont effectués sur $Y_1$ et réutilisés pour $Y_2$. La boucle de calcul pour $Y_2$ commencera donc à $i = C_{y_1,y_2} + 1$.

### 3.2.5 Le rejet sur seuil

Les candidats retenus ne doivent pas être éloignés de plus d'une distance $D_s$ de la chaîne d'entrée $X$. De façon à tenir compte de la longueur $l_y$ du mot $Y$, la valeur de $D_s$ est déterminée de la manière suivante : $D_s = l_y \times S$ où $S$ désigne la valeur du seuil par caractère. De plus, par définition, $D(i_1, j_1) \leq D(i_2, j_2)$ pour $i_1 \leq i_2$ et $j_1 \leq j_2$. Cela implique que si, pour un $i$ donné, tous les $D(i,j)$ sont supérieurs ou égaux à $i \times S$, le mot de dictionnaire en cours de traitement n'a déjà plus aucune chance de faire partie des candidats. Il peut donc être rejeté, sans calculer les $D(k,j)$ pour $k > i$.

Cette optimisation peut être poussée plus loin. Pour chaque chaîne, au plus $N_c$ candidats sont retenus. Lors du traitement d'une chaîne $X$, dès que $N_c$ mots ont obtenus un score inférieur ou égal à $D_s$, le plus mauvais des $N_c$ scores peut être considéré comme le nouveau $D_s$ pour la suite du traitement de $X$.

Le gain obtenu ici dépend fortement des chaînes d'entrée et du dictionnaire.

### 3.2.6 Estimation du gain apporté par les optimisations

Des tests ont été effectués avec différentes valeurs de $S$, en prenant les paramètres suivants :

- $r$=2 (variation maximale de longueur autorisée)

- $N_c$=10 (nombre maximum de candidats retenus)

Le dictionnaire utilisé est le dictionnaire des communes. Le test a porté sur environ 30.000 mots. Les valeurs de la table 1 représentent les facteurs gagnés en terme de temps de calcul entre :

- l'algorithme non optimisé,
- l'algorithme prenant en compte toutes les optimisations.

| $S$ | 20 | 30 | 40 | 50 | 80 | 100 |
|---|---|---|---|---|---|---|
| gain | 96 | 35 | 21 | 17 | 16 | 15 |

Table 1 : facteurs gagnés en temps de calcul avec les optimisations en fonction de la valeur de $S$ ($r$=2 et $N_c$=10)

Les gains les plus importants sont obtenus sur les mots courts. Par ailleurs, le gain entre un $S$ de 100 et un $S$ de 50 est très faible. Par contre, il est très important entre un $S$ de 50 et un $S$ de 30. Un $S$ de 50 n'est en effet pas suffisamment discriminatoire pour rejeter rapidement des mots. C'est beaucoup moins le cas d'un $S$ de 30, qui permet un élagage plus rapide. Le choix effectif du $S$ ne peut se faire uniquement au vu des résultats du tableau 1. En effet, la réduction du nombre de calculs ne doit pas pénaliser les performances de reconnaissance de l'algorithme. Des tests seront effectués pour résoudre ce problème.

## 3.3 Taux de reconnaissance du serveur de dictionnaire

Un test a été effectué sur 13.560 mots erronés appartenant à 7.600 enveloppes différentes. Les chaînes sont recherchées dans le dictionnaire des communes. Les paramètres suivants sont utilisés :

- $r$=2 (variation maximale de longueur autorisée)
- $N_c$=10 (nombre maximum de candidats retenus)
- $S$=80 (distance maximale entre la chaîne d'entrée et un candidat)

La table 2 donne les résultats obtenus. Les résultats sont inférieurs aux objectifs visés (plus de 90 % de mots corrigés). La marge de progression semble toutefois relativement grande. En effet, l'apprentissage de l'analyseur de caractères n'est pas encore complet, ce qui induit de nombreuses erreurs, notamment sur les minuscules. De plus, l'apprentissage des coûts des opérations d'édition a été réalisé sur un lot de 200 enveloppes, ce qui n'est pas suffisant pour avoir une bonne représentation de la réalité.

D'un point de vue traitement, il est évident que les méthodes les plus sophistiquées sont aussi les plus coûteuse en temps CPU. Ainsi, l'algorithme a été programmé sur un

| nombre de mots erronés traités | 13560 |
|---|---|
| % de mots corrigés | |
| premier candidat | 74,54 |
| 3 premiers candidats | 81,71 |
| % de mots non corrigés | 15,53 |

Table 2 : Taux de reconnaissance obtenus par l'algorithme ($r$=2, $N_c$=10 et $S$=80)

SUN 3/160 en C sous UNIX. La correction d'un mot de 9 lettres dure environ 9 secondes, ce qui est très loin de l'objectif de 10 ms. Pour accélérer l'algorithme, il est possible de le décomposer en tâches élémentaires permettant l'utilisation d'une architecture parallèle.

Les méthodes de programmation dynamique se prêtent bien à une mise en oeuvre sur des réseaux parallèles de processeurs, et particulièrement sur des architectures dites systoliques. La section suivante détaille l'implémentation de l'algorithme sur de telles structures.

# 4 Architecture Systolique

## 4.1 Concept Systolique

Le concept de machine systolique, dû à Kung [Kung82], consiste à obtenir, pour résoudre un problème donné, une structure régulière faite de processeurs simples et identiques, connectés suivant une topologie régulière et locale (pas de liens éloignés entre les processeurs) où les données circulent à vitesse constante. L'ensemble fonctionne par battement d'ou la désignation d'architecture systolique donné à ce type de machine.

Considérons la relation de récurrence simplifiée suivante :

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + d(x_i, y_j) \\ D(i-1,j) + d(\delta, y_j) \\ D(i,j-1) + d(x_i, \delta) \end{cases} \quad (2)$$

Une distance $D(i,j)$ se représente graphiquement comme l'intersection d'une ligne et d'une colonne d'une matrice (n x m) où n s'identifie à la longueur du mot test et m à la longueur d'un mot référence. L'idée de base est d'associer un processeur au calcul de chaque distance élémentaire $D(i,j)$ et de constituer ainsi un tableau de (n x m) processeurs comme le représente la figure 1. Le mot test (mot à corriger) est diffusé verticalement : les processeurs d'une même colonne mémorisent le même caractère. Le mot référence (mot issu d'un dictionnaire) est, quant à lui, propagé horizontalement.

Chaque processeur possède trois entrées notées EDh, EDd et EDv par lesquelles il reçoit respectivement les distances $D(i-1,j)$, $D(i,j)$ et $D(i,j-1)$ calculées par ses voisins. Il possède également une quatrième entrée (EXh) réservée à la circulation du

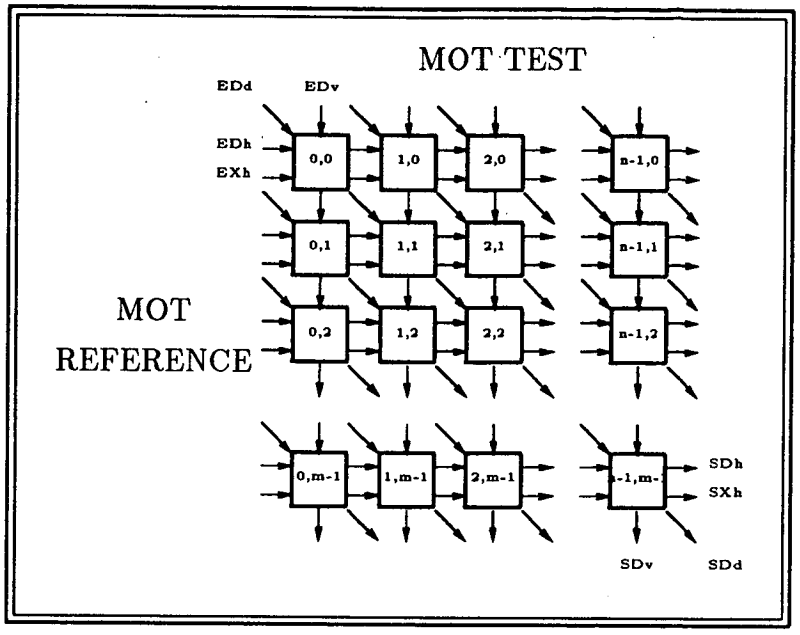


Figure 1 : tableau de processeurs

caractère du mot référence en cours de comparaison. De la même manière, quatre sorties (SDh, SDd, SDv et SXh) propagent directement les résultats aux processeurs voisins.

Le détermination de la distance entre un mot test et un mot référence s'exécute en plusieurs étapes que l'on qualifie de "cycles systoliques". Un cycle systolique se divise principalement en trois phases :

- acquisition des distances produites par les processeurs voisins,
- calcul élémentaire, en l'occurence, la minimisation des trois termes de l'équation (2),
- propagation du résultat.

La synchronisation temporel du calcul global obéit aux contraintes dûes aux dépendances engendrées par la relation de récurrence (2). Supposons le début du calcul à l'instant $t = 0$. Le processeur P(0,0) est le seul qui puisse produire un résultat puisqu'il est le seul à enregistrer des valeurs significatives sur ses entrées. Au temps $t = 1$, P(0,0) a terminé son calcul et propage $D(0,0)$ aux processeurs P(1,0), P(0,1) et P(1,1). A leur tour, P(1,0) et P(0,1) sont en mesure de calculer leur distance respective. P(1,1) ne peut commencer son calcul, il doit attendre les distances $D(0,1)$ et $D(1,0)$ générées respectivement par P(0,1) et P(1,0).

Plus généralement, au temps $t$ tous les processeurs P(i,j), tels que $t = i+j$, sont actifs. L'ensemble de ces processeurs forme une diagonale qui progresse sur le réseau à mesure que le calcul s'avance. Par conséquent, le calcul final est disponible sur la sortie SDd du

processeur P(n-1,m-1) à l'instant $t = m + n - 2$, ce qui revient à dire qu'il faut m+n-1 cycles systoliques pour obtenir une distance entre deux mots de longueur n et m.

Puisqu'une seule diagonale du réseau est active à un instant donné, les autres sont par conséquent disponibles pour le calcul de nouvelles distances. Aussi, pour l'application qui nous préoccupe où un mot particulier doit être comparé à un dictionnaire de taille importante, l'enchaînement des calculs des distances pour des mots consécutifs du dictionnaire est relativement aisé dans la mesure où ces calculs sont facilement pipelinables.

Un nouveau mot du dictionnaire est présenté à chaque cycle, mais de manière décalée dans le temps. Au temps $t = 0$, par exemple, le processeur P(0,0) traite le premier caractère du mot M0 du dictionnaire. Au temps $t = 1$ il traite le premier caractère du mot M1 du dictionnaire tandis que le processeur P(0,1) traite le second caractère du mot M0.

Dans cette approche, un résultat est disponible tous les cycles sur la sortie SDd du processeur P(n-1,m-1) et tous les processeurs travaillent de façon permanente. Le parallélisme obtenu est donc maximum.

L'architecture obtenue est une architecture régulière, où les processeurs exécutent le même algorithme de manière complètement synchrône, et conforme au concept systolique précédement introduit.

## 4.2 Structure Bidimensionnelle

L'architecture bidimensionnelle est la manière la plus immédiate et la plus directe pour implémenter des méthodes ayant trait à la programmation dynamique. Elle possède, en outre, l'incontestable avantage d'être très rapide puisqu'un résultat de comparaison est délivré à chaque cycle systolique. La durée d'un tel cycle dépend évidemment du processeur élémentaire utilisé et de l'importance du calcul à effectuer. Ces structures bidimensionnelles, dont les principaux avantages résident dans la rapidité et la régularité, présentent en contre-partie deux inconvenients qu'il convient de souligner :

Le premier est dû tout simplement au nombre de processeurs requis pour mettre en oeuvre une telle structure, l'idéal étant d'avoir un réseau de (n x m) processeurs pour comparer un mot test de longueur n avec un mot référence de longueur m. Dans ce cas, m est donné par l'estimation du nombre maximum de caractères d'un mot test tandis que n représente la longueur du mot le plus long du vocabulaire.

Il est néanmoins possible de limiter cé nombre important de processeurs en se basant sur le fait que bon nombre d'entre eux, dans notre application, n'apportent aucune contribution au résultat final, comme on l'a montré à la section 3.2.3. Les calculs significatifs sont évalués autour de la diagonale et à l'intérieur d'une bande plus ou moins étroite. Par conséquent, les processeurs en dehors de cette bande sont éliminés.

Pour un calcul sur une bande de largeur $b$ ($b = 2r + 1$) et de longueur $n$, $N$ processeurs sont nécessaires tel que :

$$N = n(2r+1) - r(r+1) \tag{3}$$

Le nombre total de processeurs est alors réduit mais reste quand même un facteur qui doit être pris en considération si la mise en oeuvre d'une telle structure est envisagée.

Le second inconvénient concerne l'alimentation en données des réseaux systoliques bidimensionnels. Il faut en effet qu'à chaque cycle, de nouvelles données soient présentées sur les entrées des processeurs périphériques. Les dispositifs fournissant les données doivent donc gérer convenablement leur séquencement et supporter le débit du réseau.

Dans notre cas, un mot référence est fourni tous les cycles systoliques, soit une valeur par processeur périphérique vertical. La "bande passante", en terme de quantité d'informations élémentaires (nombre de bits/seconde), est donnée par la formule suivante :

$$B = \frac{dnl}{t} \tag{4}$$

où

- $d$ = nombre de données à fournir pour un processeur périphérique à chaque cycle
- $n$ = dimension du tableau systolique,
- $l$ = nombre de bits nécessaires pour coder les données (coefficients d'insertion, caractères, ..),
- $t$ = durée du cycle systolique.

En pratique, cette valeur se révèle être importante et implique une interface adaptée et efficace entre le réseau systolique et le monde extérieur. Il faut donc toujours avoir à l'esprit que la réalisation d'une structure bidimensionnelle ne se réduit pas à la simple interconnexion de processeurs élémentaires, mais qu'elle doit, au contraire, s'accompagner d'un environnement performant, capable de fournir au rythme du réseau, toutes les données nécessaires.

La section suivante décrit une autre architecture systolique qui minimise ces problèmes liés au nombre de processeurs, au détriment, bien sûr, des performances globales du réseau.

## 4.3 Structure Linéaire

Le réseau systolique linéaire est une architecture directement déduite de la précédente. L'idée de base consiste à émuler une colonne du tableau par un processeur élémentaire. Le réseau obtenu est celui représenté par la figure 2.

Un processeur P(j) effectue séquentiellement les calculs des distances que produiraient des processeurs P(0,j) à P(n-1,j) sur une structure bidimensionnelle. Chaque processeur

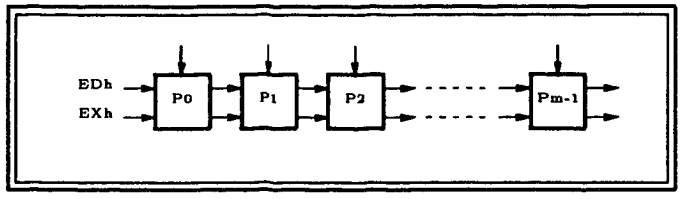


Figure 2 : Structure Linéaire

P(j) mémorise un caractère du mot test pendant que les mots référence circulent caractère par caractère d'un processeur P(j) vers un processeur P(j+1).

Chaque mot du dictionnaire est terminé par un caractère spécial, un marqueur $\perp$ , qui indique la fin du mot en cours de comparaison et le début du suivant. De cette manière, les résultats sont récupérés sur la sortie SDd du processeur uniquement lorsque le marqueur apparaît. Un processeur P(j) reçoit et transmet trois données :

- $D(i, j-1)$ : distance transmise horizontalement sur l'architecture 2D,
- $D(i-1, j-1)$ : distance transmise diagonalement,
- Xi : $i^{eme}$ caractère du mot référence.

La distance $D(i-1, j)$ qui transite verticalement sur le réseau 2D reste en permanence dans le processeur P(j).

Le temps de comparaison d'un mot du dictionnaire est directement proportionnel au nombre de lettres qui le compose. Il faut (n+1) cycles systoliques, n étant la longueur du mot référence et "+1" le marqueur de fin de chaîne, traité par les processeurs comme un caractère à part entière.

L'avantage de ce type de réseau par rapport à l'architecture bidimensionnelle réside dans le nombre plus restreint de processeurs mis en jeu. En contre-partie, le temps de réponse d'une comparaison d'un mot test avec l'ensemble d'un dictionnaire s'en trouve notablement dégradé.

Le rapport des temps de réponses (noté $\alpha$) entre les deux architectures est donné par :

$$\alpha = \frac{N + C}{N} \tag{5}$$

avec

- $N$ = nombre de mots du dictionnaire
- $C$ = nombre de caractères du dictionnaire

Pour illustrer notre propos, le dictionnaire des communes françaises ($\approx$ 30000 mots) donne un rapport d'environ 8,5. Globalement et pour cette application, un réseau linéaire

est donc environ 8,5 fois moins performant (en moyenne) qu'une structure 2-D, mais le gain en processeur est grandement appréciable.

L'optimisation qui consiste à n'effectuer que les calculs nécessaires (bande diagonale sur un réseau 2D) s'applique également au réseau linéaire. L'algorithme de chaque processeur est cependant légèrement différent. Contrairement à l'émulation "pure et dure" d'une colonne de processeurs d'un réseau 2-D, avec le respect du séquencement des données qui en découle (émulation de P(0,j), envoi des données à P(j+1), émulation de P(1,j), envoi des données, ..etc..), chaque processeur acquiert en bloc toutes les données dont il a besoin, effectue son calcul sur une colonne réduite à la largeur de la bande considérée, puis propage l'ensemble des résultats au processeur suivant.

Un cycle systolique est un ensemble de $b$ sous-calculs ($b$ est la largeur de la bande) dont chaque durée est équivalente à la durée du cycle systolique précédent. Un résultat est délivré à chaque nouveau cycle systolique ainsi défini. Le choix de la largeur de bande $b$ donne directement le rapport de performance entre l'architecture 2-D et la structure linéaire.

# 5 Evaluation des Architectures Systoliques

Dans cette dernière partie, différentes architectures sont évaluées à partir de critères tels que le type de processeurs utilisés, les performances globales des réseaux considérées, ou encore, la complexité de l'interface nécessaire au bon fonctionnement du réseau.

Par la suite, il est supposé que tous les processeurs, quels que soient leur nature et l'architecture à laquelle ils appartiennent, exécutent le même algorithme de base, soit le calcul de la distance simplifiée :

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + d(x_i, y_j) \\ D(i-1,j) + 100 \\ D(i,j-1) + 100 \end{cases} \quad (6)$$

## 5.1 Architecture dédiée

Une première solution consiste à considérer que l'algorithme est complètement figé. Dans ce cas, la fonction affectée à chaque processeur est directement cablée. La mise en oeuvre matérielle s'aiguillera, soit vers une solution classique à base de boitiers standard du commerce, soit vers la mise au point d'un VLSI spécialisé. En effet, la fonctionnalité d'un processeur élémentaire est relativement peu complexe et peut, par exemple, être realisé à l'aide d'opérateurs agencés de la manière représentée par la figure 3.

Le temps de cycle d'une tel processeur est fonction de la rapidité des opérateurs de base. La solution la plus simple et la plus immédiate consiste à les interconnecter directement; dans ce cas, la durée du calcul est égal à la somme des temps de traversée des opérateurs situés sur le chemin le plus long entre les entrées et la sortie.

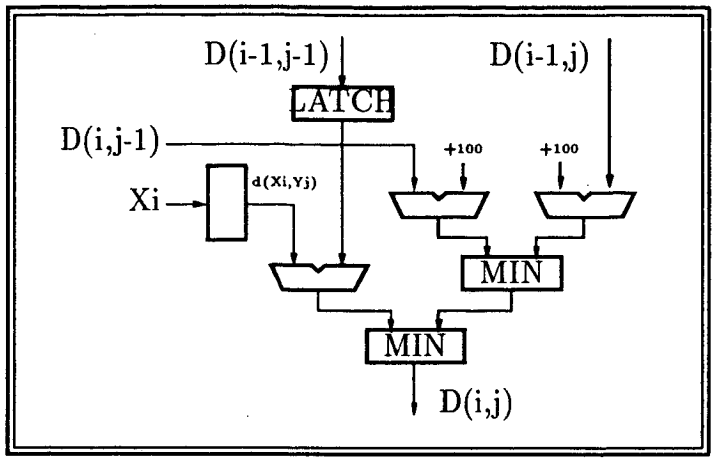


Figure 3 : processeur élémentaire cablé

### 5.1.1 Structure 2-D

En version bidimensionnelle, cette approche permet, a priori, de traiter plusieurs dizaines de millier de mots en 10ms, indépendamment du schéma retenu (tableau complet ou tableau réduit à une diagonale de largeur $b$). De plus, la nature pipeline du réseau 2-D est appliquable à la structure interne de chaque processeur élémentaire dans la mesure où un opérateur de base peut être divisé en étages successifs correspondant à un niveau de pipeline. Le temps de cycle devient alors égal au temps de transfert de l'opérateur le plus lent. L'exemple de la figure 3 donne une structure équivalente à un niveau de pipeline égal à 3.

### 5.1.2 Structure 1-D

La mise en oeuvre d'une structure 1-D est beaucoup moins immédiate qu'un réalisation sur structure 2-D. La circulation caractère par caractère d'un mot référence à travers le réseau oblige l'ajout d'un marqueur spécial pour séparer explicitement deux mots du dictionnaire. La rencontre de ce marqueur indique au processeur l'arrivée d'un nouveau mot et est l'occasion d'une réinitialisation locale. Un dispositif "hardware" doit être capable de gérer ce type de contrainte. D'autre part, le pipeline des opérateurs de base n'est plus possible tel qu'il est défini sur une structure 2-D. En effet, un processeur P(j) émulant successivement les unités d'une même colonne a besoin, avant d'entamer le calcul d'une distance $D(i,j)$, de la distance $D(i-1,j)$. Il est cependant possible de conserver l'architecture pipeline interne en introduisant la notion d'entrelacement des calculs. L'idée est que, pour un processeur possédant un niveau de pipeline de $p$ étages, $p$ calculs (liés à des références distinctes) se déroulent en même temps, mais à des niveaux d'avancement différents. Le processus d'alimentation du réseau est alors beaucoup plus complexe et la récupération des résultats beaucoup plus délicate.

L'optimisation relative à l'émulation d'une bande de $b$ processeurs et se traduisant par un cycle systolique équivalent au calcul de $b$ distances par processeur et par mot est encore plus complexe à mettre en oeuvre sur une architecture complètement figée. Chaque processeur reçoit (théoriquement) en bloc, toutes les informations utiles pour l'ensemble de son calcul puis propage l'ensemble de ses résultats. S'il est, a priori, possible de séquencer dans le temps les différentes étapes de son calcul il subsiste néanmoins un problème lié au nombre d'informations non constantes qui transitent sur le réseau. En effet, la comparaison d'un mot référence de longueur $n$ demande le transfert des $n$ caractères, alors que le calcul de base ne nécessite que $b$ caractères, mais différents d'un processeur à l'autre. Cette dissymétrie entre le nombre de calculs constant et la variation du nombre de données qui transitent se prête assez mal à une réalisation purement cablée.

## 5.2 Architecture à base de processeurs spécialisés

Une autre alternative pour réaliser un réseau systolique consiste à utiliser pour processeurs élémentaires des éléments non spécifiques à l'application envisagée mais présentant néanmoins de bonnes dispositions pour l'intégration dans un réseau de ce type, notamment au niveau des interconnexions et communications avec les processeurs voisins. Ces processeurs, n'étant pas dévolus à une application particulière, sont micro-programmables et, par conséquent, accroissent notablement la durée du cycle systolique de base.

Un tel processeur, appelé API15C, a été développé à l'IRISA. C'est un circuit VLSI CMOS micro-programmable d'une complexité d'environ 50.000 transistors. Il possède trois ports parallèles d'entrée/sortie sur 16 bits. Cette structure permet d'utiliser ce composant aussi bien dans une structure bidimensionnelle que linéaire [Fris88a][Fris88b]. L'architecture interne du circuit comporte différents modules : deux blocs de 16 registres, une unité arithmétique et logique, une mémoire de 256 mots et un multiplieur entier sur 16 bits.

### 5.2.1 Structure 2-D

Par rapport à la version cablée, il n'existe aucun changement de principe si ce n'est que la notion de pipeline n'a, ici, aucun sens. Pour donner un ordre de grandeur, le calcul de l'équation (6) sur le processeur API15C dure environ 2 $\mu$s. Si cette solution est nettement moins rapide que la précédente, elle possède néanmoins l'avantage non négligeable d'une remise en cause possible, et à tout instant, de l'algorithme utilisé.

La réalisation du dispositif d'alimentation en données d'un réseau à base de processeurs de ce type devient alors plus complexe dans la mesure où on se réserve une marge de sécurité quant à l'aspect définitif de l'algorithme. Le séquencement des données est susceptible d'être modifié à tout instant et ne doit en aucune manière remettre en cause l'architecture établie. A l'image des processeurs élémentaires l'ensemble du dispositif d'alimentation du réseau doit être programmable.

### 5.2.2 Structure 1-D

La micro-programmation des processeurs élémentaires facilite grandement l'implémentation de l'algorithme de correction de fautes sur une architecture linéaire. Les problèmes dûs à la spécification d'un caractère (le marqueur) ou à la variation des données qui transitent par le réseau sont directement traités par programmation. Cette solution est néanmoins, de loin, la moins satisfaisante en terme de rapidité de calcul.

## 5.3 Performances des Architectures

Cette section tente d'évaluer les performances des architectures systoliques proposées. Le tableau ci-dessous donne une idée réaliste de la quantité d'informations qu'il est possible de traiter suivant la solution choisie.

| ARCHITECTURES | *processeurs dédiés* t = 100ns | *processeurs spécialisés* t = 2$\mu$s |
|---|---|---|
| 2-D pipeline | 100.000 | - |
| 2-D normal | 33.000 | 5.000 |
| 1-D pipeline entrelacé | 11.760 | - |
| 1-D optimisé | 6.600 | 1.000 |
| 1-D normal | 3.900 | 600 |

Les valeurs du tableau indiquent le nombre de mots traités en 10 ms suivant l'architecture et le type de processeur désirés et en se limitant aux calculs autour d'une diagonale de 5 processeurs. La version dédiée suppose des opérateurs de base avec un temps de transfert de 100ns tandis que les processeurs spécialisés sont supposés calculer l'équation de base en un cycle systolique de durée égale à 2$\mu$s

Le choix d'un mot test de 7 caractères a été retenu parce qu'il correspond à la longueur de mot demandant le plus de calcul. En effet, les communes de 7 caractères étant les plus nombreuses, celles comprises entre 5 et 9 caractères constituent la partition du dictionnaire la plus importante, soit environ 20.000 mots.

Enfin, le temps de réponse du serveur de correction ne doit pas être basé sur le temps de réponse moyen mais sur le temps maximum. Le tri du courrier exige un cadencement régulier et les lettres disposent d'un durée limitée pour être aiguillées vers leur destination, quel que soit le nombre d'erreurs qu'elles contiennent. La solution systolique répond parfaitement à cette exigence dans la mesure où on connait à l'avance le temps de réponse du système.

Parmi les architectures proposées, seule la solution bidimensionnelle à bases de processeurs cablés répond aux exigences des objectifs fixés, à savoir, traiter 20.000 mots

en moins de 10 ms. Les processeurs élémentaires étant relativement peu complexes, l'intégration de plusieurs éléments sur une même puce offre l'avantage de constituer un réseau performant, rapide et de faible encombrement.

On peut remarquer également que la solution bidimensionnelle à base de processeurs spécialisés atteint de très bonnes performances. Un facteur 4 la sépare des objectifs. Cependant, on peut imaginer que ce facteur peut être réduit à court terme grâce aux progrès technologiques.

Les solutions linéaires, bien qu'en deça des objectifs, atteignent cependant des performances intéressantes. A titre d'exemple, la solution 1-D optimisée à base de processeurs spécialisés permet le traitement de 20.000 mots en 200 ms, soit un gain de 45 par rapport à la version sur un SUN 3/160 (cf paragraphe 3.3).

Notons enfin que les architectures systoliques proposées ne tiennent que partiellement compte des optimisations de l'algorithme présentées au paragraphe 3.2, puisque seule l'optimisation des diagonales est utilisée. En effet, la structure régulière du traitement des données sur ces architectures s'y prête mal. L'implémentation des optimisations doit être étudiée principalement sur les architectures linéaires, afin d'améliorer leurs performances.

# 6 Conclusion

Nous avons décrit un algorithme de correction de chaînes de caractères permettant de corriger les mots d'une adresse. Cet algorithme, basé sur le concept de métrique de Levenshtein, donne des taux de correction prometteurs, bien meilleurs que ceux obtenus par les méthodes actuellement utilisées pour traiter des libellés d'adresses. Il pose toutefois le problème du temps de calcul, notamment sur une machine séquentielle, du fait de la nature temps réel de l'application.

Pour résoudre ce problème, nous avons étudié différentes mises en oeuvre possibles sur des réseaux systoliques linéaires ou bidimensionnels, soit complètement dédiés, soit plus généraux. Les solutions les plus rapides sont obtenues sur des réseaux bidimensionnels complètement dédiés. De tels réseaux permettent d'atteindre, et même de dépasser les objectifs temporels visés. Ils nécessitent toutefois une gestion de données complexe et n'autorisent pas de modifications de l'algorithme.

Les réseaux linéaires, bien que donnant des performances assez loin des objectifs, permettent cependant d'atteindre de très bonnes performances par rapport à des solutions classiques (programmées).

L'étude doit être complètée par la mise en oeuvre des algorithmes sur des architectures parallèles plus générales (hypercube, architecture à base de transputer). Les communications étant relativement coûteuses sur ce type de machines, la méthode de parallélisation la mieux adaptée est sans doute différente de celle utilisée pour les architectures systoliques. Il paraît en effet plus judicieux de partager le dictionnaire entre les différents processeurs. Chacun détermine alors des candidats dans la partie de dictionnaire dont il

dispose. La synthèse des résultats se fait, soit globalement sur un processeur hôte, soit localement par sous-groupes de processeurs.

## Remerciements

Nous tenons à remercier F. Bodin pour son active collaboration, notamment, sur les optimisations algorithmiques qui ont été proposées sur l'architecture linéaire.

Nous remercions également E. Gautrin pour la lecture attentive de cet article et des nombreuses remarques qui, nous l'espérons, contribuent à la clarté et à la compréhension de ce texte.

## Bibliographie

[Ange83] R.C Angell, G. E. Freund, P. Willett, "Automatic spelling correction using a trigram similarity measure," *Infor. Proc. and Manag.*, vol. 19, n° 4, pp. 225-261, 1983.

[Bick87] M. A. Bickel, "Automatic correction to misspelled names : a fourth-generation language approach," *Comm. ACM,* vol. 30, n° 3, pp. 224-228, 1987.

[Blai60] C. R. Blair, "A program for correcting spelling errors," *Inform. Contr.,* pp. 60-67, 1960.

[Char86] F. Charot, P. Frison, P.Quinton, "Systolic Architectures for Connected Speech Recognition," *IEEE Trans on ASSP,* vol. 34, n° 4, pp. 765-779, 1986.

[Dame64] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM,* vol. 7, pp. 171-176, 1964.

[Davi62] L. Davidson, "Retrieval of misspelled names in an Airlinés Passenger record system," *Comm. ACM,* vol. 5, n° 3, pp. 169-171, 1962.

[Durh83] I. Durham, D. A. Lamb, J. B. Saxe, "Spelling correction in user interfaces," *Comm. ACM* vol. 26, n° 10, pp. 764-773, 1983.

[Fris85] P. Frison, P. Quinton, "An Integrated Systolic Machine for Speech Recognition", in *VLSI : Algorithms and Architectures,* Edited by P. Bertolazzi and F. Luccio, North Holland, pp. 175-186, 1985.

[Fris88a] P. Frison, D. Lavenier, "A Fast Machine for Prototyping String Correction Algorithms", *RIAO 88: Users-Oriented Content-Based Text and Image Handling,* MIT, Cambridge, USA, Mars88.

[Fris88b] P. Frison, D. Lavenier, "A VLSI Systolic Machine for String Correction", *ICS 88: Third International Conference on Supercomputing,* Boston, MA, May 1988.

[Hall80] P. A. V. Hall, G. R. Dowling, "Approximate string matching," *Comput. Surv.,* vol. 12, pp. 381-402, 1980.

[Kash84] R. L. Kashyap, B. J. Oommen, "Spelling correction error using probabilistic methods," *Computer text recognition and error correction,* ed. by S. Srihari, IEEE Computer society press, pp. 272-279, 1984.

[Kung82] H. T. Kung, "Why systolic architectures," *Computer,* vol. 15, pp. 37-46, Jan. 1982.

[Lowr75] R. Lowrance, R. A. Wagner, "An extension of the string to string correction problem," *J. Assoc. Comput. Mach.,* vol. 22, n° 2, pp. 177-183, 1975.

[Morr75] R. Morris, L. L. Cherry, "Computer detection of typographical errors," *IEEE transaction on professional communications,* vol. PC-18, n° 1, pp. 54-64, 1975.

[Muth77] F. Muth, A. L. Tharp, "Correcting human error in alphanumerical terminal input," *Information Proc. and manage,* vol. 13, n° 6, pp. 329-337, 1977.

[Okud76] T. Okuda, E. Tanaka, T. Kasai, "A method for correction of garbled words based on the Levenshtein metric," *IEEE Trans. Comput.,* vol. C-25, pp. 172-177, 1976.

[Oomm87] B. J. Oommen, "Recognition of noisy subsequences using constrained edit distances," *IEEE Trans. PAMI* vol. 9, n° 5, pp. 676-685, 1987.

[Pete80] J. L. Peterson, "Computer programs for detecting and correcting spelling errors," *Comm. assoc. comput. mach.,* vol. 23, pp.676-687, 1980.

[Vero87] J. Veronis "Phonographic versus typographic correction in natural language interfaces," *Fifth int. Symp. in Applied Informatics,* GrindelWald, Switzerland, Feb. 87.

[Wagn74] R. A. Wagner, M. J. Fisher, "The string to string correction problem," *J. ACM,* vol. 21, pp. 168-173, 1976.

[West83] N. Weste, D. J. Burr, B. D. Ackland, "Dynamic time warp pattern matching using an integrated multiprocessing array," *Computer,* vol. 15, pp. 37-46, 1982.

[Yode82] M. A. Yoder, L. J. Siegel, "Dynamic time warping algorithms for SIMD machines and VLSI processor arrays," in *Proc. 1982 Int. Conf. Acoust., Speech, Signal Processing,* Paris, France, May 1982, pp. 1274-1277.

## LISTE DES DERNIERES PUBLICATIONS INTERNES

**PI 449** **CONVOLUTION SYSTOLIQUE DE FONCTIONS ARITHMETIQUES**
Patrice QUINTON, Yves ROBERT
30 Pages, Janvier 1989.

**PI 450** **MISE EN OEUVRE D'ALGORITHMES NUMERIQUES SUR UN HYPERCUBE**
Brigitte VITAL
28 Pages, Janvier 1989.

**PI 451** **LANCER DE RAYON SUR DES ARCHITECTURES PARALLELES : UNE ETUDE DE PERFORMANCE**
Thierry PRIOL, Kadi BOUATOUCH
20 Pages, Janvier 1989.

**PI 452** **LANCER DE RAYON : APPROCHES PARALLELES**
Didier BADOUEL, François BODIN, Thierry PRIOL
16 Pages, Janvier 1989.

**PI 453** **UN COMPILATEUR ESTELLE MULTI-PROCESSEURS POUR L'EXPERIMENTATION D'ALGORITHMES DISTRIBUES SUR MACHINES PARALLELES**
Jean-Marc JEZEQUEL, Claude JARD
54 Pages, Janvier 1989.

**PI 454** **REALISATION ET CALIBRATION D'UN SYSTEME EXPERIMENTAL DE VISION COMPOSE D'UNE CAMERA MOBILE EMBARQUEE SUR UN ROBOT-MANIPULATEUR**
François CHAUMETTE, Patrick RIVES
36 Pages, Février 1989.

**PI 455** **ARCHITECTURE SYSTOLIQUE POUR LA CORRECTION AUTOMATIQUE DE LIBELLE D'ADRESSE**
Dominique LAVENIER, Jean-Luc SCHARBARG, Patrice FRISON
22 Pages, Février 1989.



Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique